1. **Introducción**

En este taller se va a experimentar con modelos colaborativos para sistemas de recomendación utilizando un *dataset*de recomendación de música obtenido en <https://www.dtic.upf.edu/~ocelma/MusicRecommendationDataset/lastfm-1K.html>.

1. **Conocimiento del *dataset*de trabajo**

El *dataset* incluye la información de los hábitos de reproducción de los usuarios de *Last.fm[1]* de cerca de 1000 usuarios hasta el 5 de mayo del 2009, e información sobre los usuarios en dos tablas separadas: *userid-profile* y *userid-timestamp-artid-artname-traid-traname.tsv*.

Las entradas de datos de *userid-timestamp-artid-artname-traid-traname.tsv*relacionan un usuario con una fecha y hora (una marca temporal o un *timestamp*), un artista y una canción. El formato en el que están los datos es .tsv. La organización por columnas es <id del usuario, *timestamp*, id del artista, id de la canción>, donde el identificador tanto del artista como de la canción es un MBID.  Un MBID significa *MusicBrainz Identifiers*y cada MBID es único para una canción o un artista. La siguiente tabla muestra un breve perfilamiento de los datos:

|  |  |  |
| --- | --- | --- |
| Columna | ¿Contiene nulos? | Tipo de valores |
| userId | No | Cadena de texto |
| timestamp | No | Cadena de texto |
| artistId | Sí | Cadena de texto |
| artistName | Sí | Cadena de texto |
| trackId | Sí | Cadena de texto |
| trackName | Sí | Cadena de texto |

Tabla 1: Breve perfilamiento de datos

La primera decisión que se tomó al preprocesar los datos es eliminar las filas que contengan nulos en alguno de los identificadores, ya que es lo que se utiliza para generar el sistema de filtro colaborativo. En total se eliminaron XXXX filas por este motivo. También se eliminaron las filas que presentaran problemas de formato (más separadores de los necesarios, etc.). 9 filas se eliminaron de esta manera.

Esta tabla no incluye un *rating*del usuario a la canción o al artista, es únicamente información de hábitos de reproducción; un ejemplo se muestra a continuación.

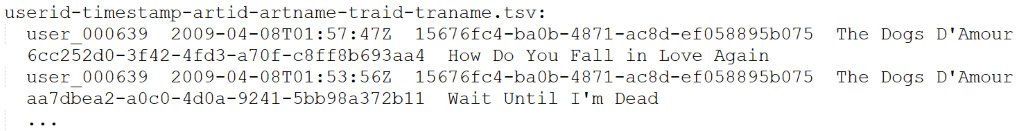


Figura 1. Ejemplo de fila de los datos en *userid-timestamp-artid-artname-traid-traname.tsv*.

Hay 83905 artistas diferentes que para efectos del modelo de sistemas de recomendación, son los ítems a recomendar.

La información de la tabla en *userid-profile*incluye información de los usuarios y un breve perfilamiento de ellos. Incluye el *id*del usuario, género, edad, país y fecha de registro. A continuación, se muestra el perfilamiento de estos datos y un ejemplo:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Columna | ¿Contiene nulos? | Número de valores no nulos | Número de valores únicos | Tipo de valores |
| #id | No | 992 | 992 | Valores consecutivos desde user\_000001 hasta user\_001000, con algunos faltantes |
| gender | Sí | 884 | 2 | M o F (o vacío) |
| age | Sí | 286 | 34 | Número entre 3 y 103, con algunos faltantes y vacíos |
| country | Sí | 907 | 66 | Nombre del país. Hay varios países que solo tienen un usuario, 25% de los usuarios son de Estados Unidos, 14% del Reino Unido, 5,5% de Polonia. El resto están todos debajo de 4%. |
| registered | Sí | 984 | 625 | Fecha entre el 29 de octubre del 2002 y el 5 de noviembre del 2009, con algunos faltantes. |

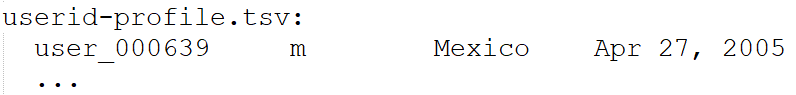


Figura 2. Ejemplo de fila en los datos en *userid-profile*

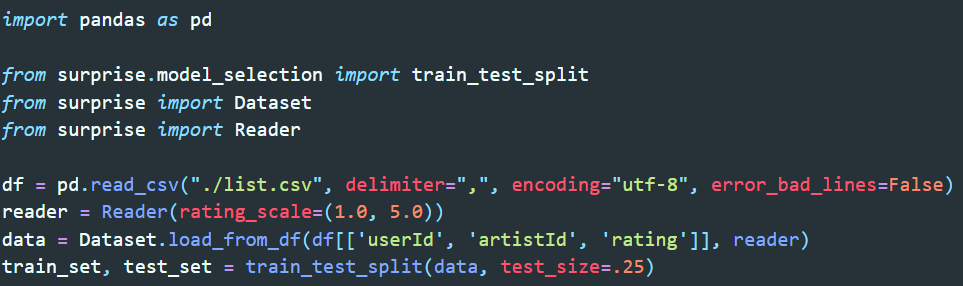
1. **Pre-procesamiento de datos**

Como no se tienen *ratings,*se van a definir según la interacción que tenga un usuario con los artistas. Para ello, se definió utilizar una función logarítmica que incluye el artista que más fue escuchado, de la forma: RatingArtista = LogMaxReproducido(ReproduccionesArtista) \* 4 + 1, si ReproduccionesArtista no es 0; en caso contrario, dejar este valor como 0 (esto ocurre cuando el usuario nunca ha escuchado al artista).

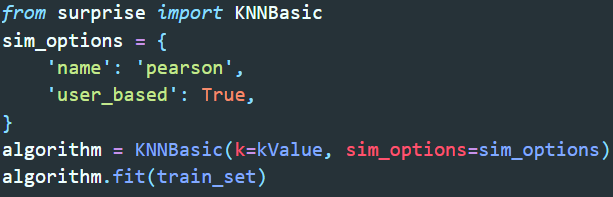
Esta fórmula devuelve un número entre 1.0 y 5.0. Se decidió así para no tener un comportamiento lineal que se vea sesgado a un artista con muchas reproducciones. Por ejemplo, usuario a escuchó el artista x un total de 1000 veces, y escuchó al grupo y un total de 50 veces. Si bien 50 reproducciones son relevantes, en una escala lineal se le daría un rating muy bajo.

Luego se formó una matriz donde cada columna es un usuario y cada fila un artista, y el contenido de la tabla es el rating que le da un usuario a un artista dado. Dado que esta matriz contiene muchos valores en cero, se formó una nueva matriz de tres columnas *userId*, *artistId*, *rating*, que reduce considerablemente el espacio necesario para almacenarla y sirve como entrada para un sistema de recomendación en la librería *Surprise*.

Los datos se separaron en dos conjuntos luego del pre-procesamiento, 25% de los datos se reservaron como *dataset*de pruebas. Esta separación se realiza con las siguientes instrucciones en *Surprise*, específicamente ***surprise.model\_selection.train\_test\_split***.



1. **Construcción de modelos colaborativos usuario usuario**
2. El modelo se construyó utilizando la librería surprise de Python. Para construir el modelo, se debe definir un algoritmo de predicción, en nuestro caso KNNBasic. KNNBasic recibe como parámetros el número máximo de vecinos, el nombre de una función de similitud y si es usuario-usuario o ítem-ítem.



1. Para predecir la relevancia de los ítems se utiliza la función *test*.

C:\Users\Rogelio Garcia\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D1D59B0A.tmp

Esta función arroja predicciones con el identificador de usuarios *uid*, identificador del ítem *iid*, el *rating*real y el estimado, y otras informaciones. También incluye un campo para saber si fue posible predecir el rating para el ítem dado.

Prediction(uid='user\_000500', iid='2870dffb-f336-4a7b-96c0-eecf417c7fed', r\_ui=1.0, est=1.8971555182565383, details={'was\_impossible': True, 'reason': 'User and/or item is unkown.'})

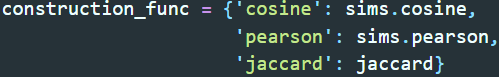
Prediction(uid='user\_000722', iid='d4a1404d-e00c-4bac-b3ba-e3557f6468d6', r\_ui=3.049423276345537, est=1.886442260279148, details={'actual\_k': 5, 'was\_impossible': False})

Adicionalmente, se definió una función que recibe las predicciones y un número entero n y retorna un diccionario que contiene una lista con las primeras n predicciones para cada usuario en orden del rating predicho, en forma de tupla (*itemId*, *rating*).

'user\_000463': [('32b90c92-9978-4a07-90eb-caa4b22f4907', 3.5106869703792136), ('d4d17620-fd97-4574-92a8-a2cb7e72ce42', 3.464256641763165), ('9de8f66e-3cd1-4f11-8328-38200f0612b0', 3.1446878961364217), ('cc0b7089-c08d-4c10-b6b0-873582c17fd6', 2.9901818013363446), ('f181961b-20f7-459e-89de-920ef03c7ed0', 2.728503442821299), ('8c538f11-c141-4588-8ecb-931083524186', 2.6670920880573963), ('970fb29f-e288-403e-a388-d2a7889bfa47', 2.643524734521892), ('8434409e-baa9-4e12-b4aa-566a91c7d7cf', 2.480210102296422), ('4efa55ba-93cf-497f-baf3-2ca9da7e193e', 2.4473736580596714), ('77f049ad-f469-4ad1-8283-7a2606a6722e', 2.436338420952059)]

Para implementar los tres índices de similitud, y para algunas partes de este taller, se volvió a implementar el modelo *KNNBasic*. La clase *KNNBasic* hereda de *SymmetricAlgoJaccard*, que a su vez hereda de *AlgoBase*. Para poder solucionar varios puntos del taller, se implementó *KNNBasic* con una ligera modificación para retornar vecinos según un umbral, y se implementó *SymmetricAlgoJaccard* con una modificación que anula o sobreescribe la función *compute\_similarities*, que retorna una matriz de similitud usuario-usuario (o ítem-ítem).

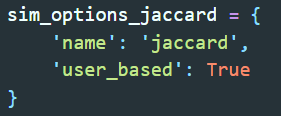
Todos los modelos utilizados son creados con la clase *KNNBasic* con un parámetro *sim\_*options diferente, que se traduce a una implementación diferente de *compute\_similarities*. Esta función utiliza otra función que calcula la matriz de similitud entre cada usuario (o cada ítem). Coseno y Pearson son importados de *similarities*, mientras que jaccard fue implementado en *jaccard\_sim.py*.



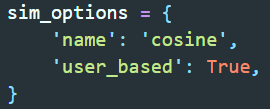
1. Para el índice de Jaccard fue necesario definir una nueva función de similitud. Esta similitud se calcula como:

De esta manera, durante el cálculo solo se mantiene una cuenta del número de artistas a los que ha calificado el usuario y la cuenta del número de artistas que los dos usuarios.

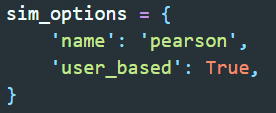
Al crear el modelo, se utiliza el siguiente parámetro:



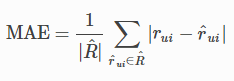
1. Un modelo basado en distancias coseno es fácil de lograr con *Surprise*. El parámetro *sim\_options* que entra como parámetro en *KNNBasic* lo define con la opción name.

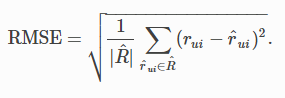


1. Un modelo basado en correlación de Pearson también es fácil de lograr con *Surprise*:

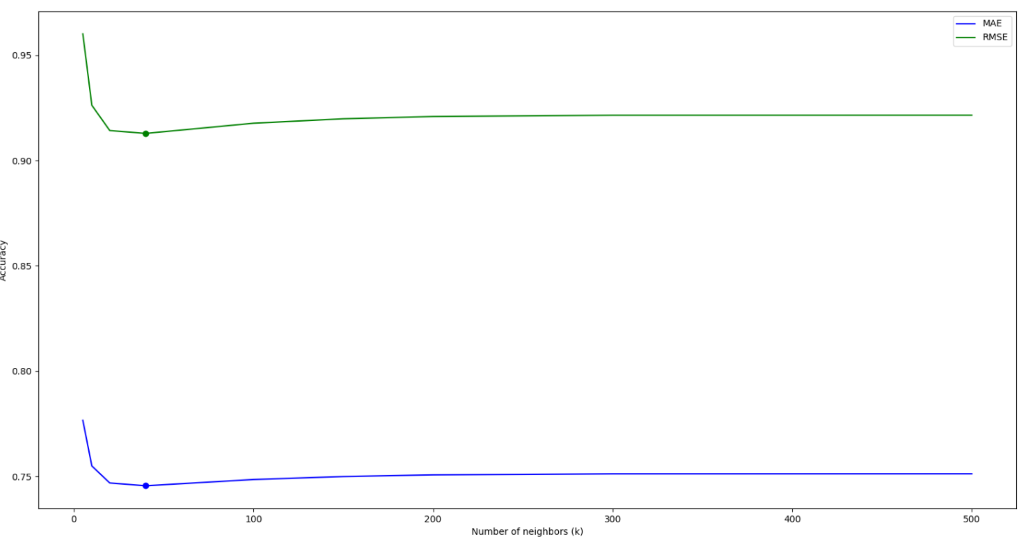


1. Para evaluar el comportamiento del modelo y cómo se está ajustando a los datos, se utilizaron las métricas RMSE (raíz del error cuadrático medio) y MAE (error absoluto medio).



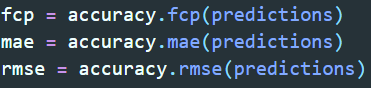


Experimentalmente se encontró que utilizando cualquiera de estas métricas los valores óptimos no varían demasiado. La siguiente figura muestra la determinación del número de vecinos óptima para el caso de la función de similitud de Pearson; varía en menos de 5 vecinos el valor óptimo.

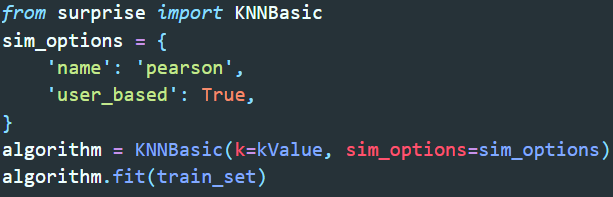


Conceptualmente, RMSE castiga más a un modelo por contener predicciones muy alejadas de los valores reales. Como no se quiere recomendar un ítem demasiado alejado de los gustos de la persona, se optó por utilizar RMSE para evaluar los diferentes modelos colaborativos.

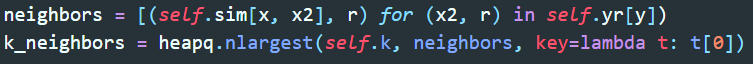
En *Surprise*se pueden calcular tres métricas diferentes: FCP (fracción de parejas concordantes), MAE y RMSE. Cada una se puede calcular así:



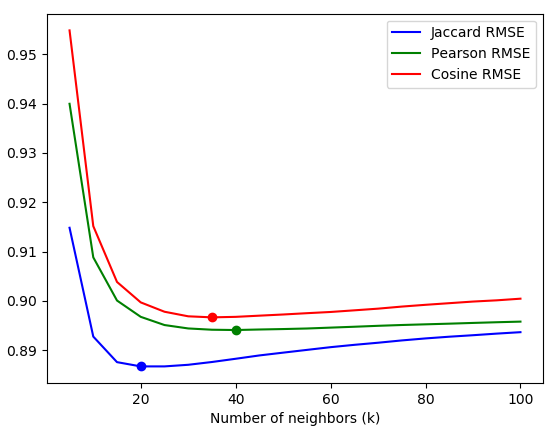
1. Se diseñó una estrategia para variar el parámetro del número de vecinos al definir el algoritmo a utilizar. En este caso es el valor *kValue*. Se itera sobre este valor y se analizan los resultados cada vez, guardando el valor para *kValue* que menor error obtuvo, por cada algoritmo analizado.



En la función *estimate*, de la clase *KNNBasic*, se forma un montículo con todos los vecinos y se retorna los *k* primeros, de la siguiente manera:



Se experimentó con un arreglo de *k\_*values de 5 a 100, en intervalos de 5. Para los tres índices de similitud utilizados hay un valor óptimo para el número de vecinos en el que el error es mínimo. A mayor o menor número de vecinos, el error aumenta. Graficamos el número óptimo de vecinos, y para los tres algoritmos está entre 20 y 40, con el menor error siendo de Jaccard.

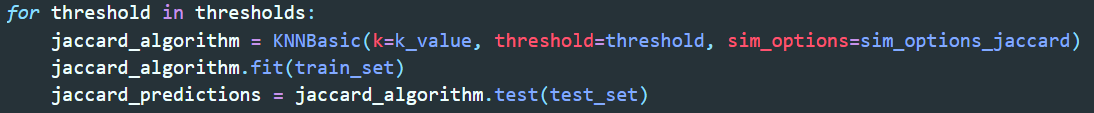


En la siguiente table se resumen los datos de varias ejecuciones y sus valores. En cada ejecución el test set cambia, por lo que los resultados varían. Sin embargo, no son muy diferentes.



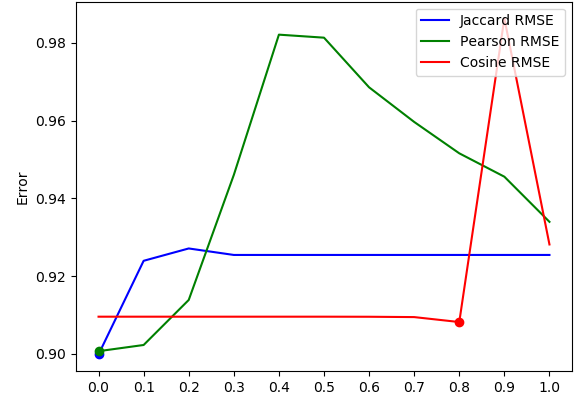


También se definió una estrategia para analizar los resultados por umbral de similitud. Para ello, en *KNNBasic* se definió un nuevo parámetro opcional *threshold* con valor de 0 por defecto, y al seleccionar los vecinos de un usuario en la función *estimate*, utilizará solo la lista de los vecinos cuya similitud supere un cierto umbral:

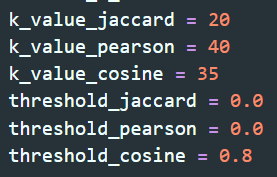


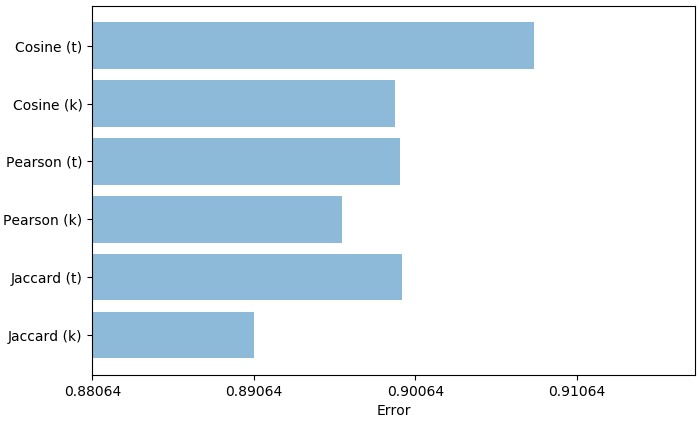


Se experimentó con un arreglo de thresholds entre 0 y 1 en intervalos de 0.1 y se graficó el resultado. El resultado se graficó de la misma manera, mostrando a Jaccard con el menor error:



Como se encontró que teniendo un umbral de similitud entre usuarios de 0.0, es decir, aceptando todos los vecinos, se tuvo mejor precisión en el modelo, se realizó un experimento entre los mejores parámetros encontrados para número de vecinos y para umbral.



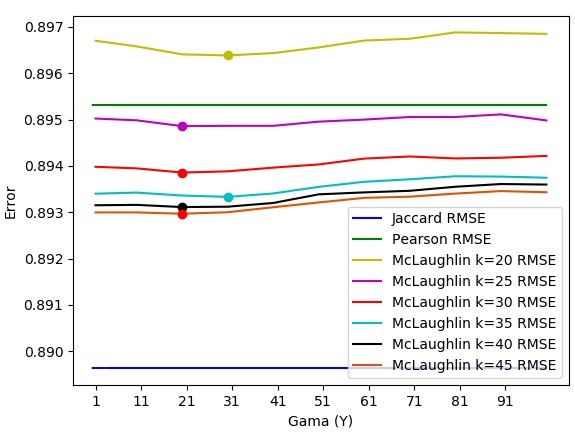


En esta gráfica se compararon los modelos con los mejores parámetros para número de vecinos y para umbral de similitud. El experimento fue realizado reiteradamente y de manera consistente, Jaccard con k=20 fue el modelo con menor error.

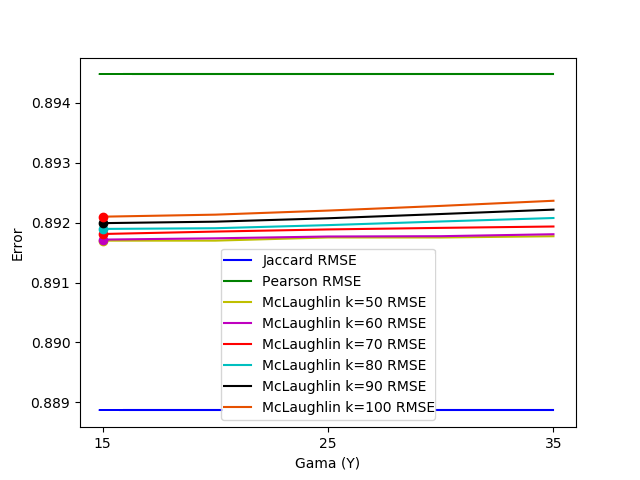
1. La fórmula de McLaughlin fue implementada de la misma manera en que se implementó Jaccard. Se definió una nueva forma de calcular la similitud, agregando la corrección de Herlocker y colaboradores,

que incluye una corrección cuando los ítems que han calificado dos usuarios en común es mayor a un valor gama definido. Esto hace que estos usuarios vean su similitud aumentada por el hecho de haber calificado ítems en común. El valor de gama óptimo se debe definir mediante experimentación, y como McLaughlin sigue siendo un algoritmo que hereda de *KNNBasic*, también es necesario definir un valor óptimo para k. Se optó por experimentar únicamente con el número de vecinos y no con un umbral dado que en el experimento anterior se observó que variar el número de vecinos logra un error menor.

El primer experimento fue variando el valor de k entre 20 y 45 en intervalos de 5, y gamma entre 1 y 100 en intervalos de 10. Jaccard y Pearson también se graficaron para k=20 y k=35 respectivamente, que son los valores óptimos encontrados en el punto 3.d.



Se puede observar que el error disminuye a medida que aumenta el valor de k. Para k=45, el gamma óptimo está entre 20 y 30. Se repitió con valores de más de 45 para descartar que siguiera disminuyendo el error. En este caso solo se graficó McLaughlin para valores de 50 hasta 100 en intervalos de 10.



Se puede observar que el mínimo error lo tiene McLaughlin con k=50, y a partir de este punto aumenta.

El paper de McLaughlin también habla de que utilizar la métrica MAE no es óptimo dado que puede que algunos errores grandes se escondan con la métrica, resultando en un MAE pequeño pero errores graves. En este caso se está utilizando RMSE, que es similar a MAE pero castiga precisamente los valores muy alejados del real. La métrica que se propone es tomar en cuenta la fracción de ítems bien catalogados. En *Surprise* se incluye una métrica, llamada FCP — *Fraction of Concordant Pairs*, que toma en cuenta si un ítem se catalogó bien **en relación** a los demás ítems. Se utilizó FCP para comprar los resultados dado que esta métrica busca evitar tener en cuenta únicamente la distancia del valor predicho al valor real, como lo hacen MAE y RMSE.

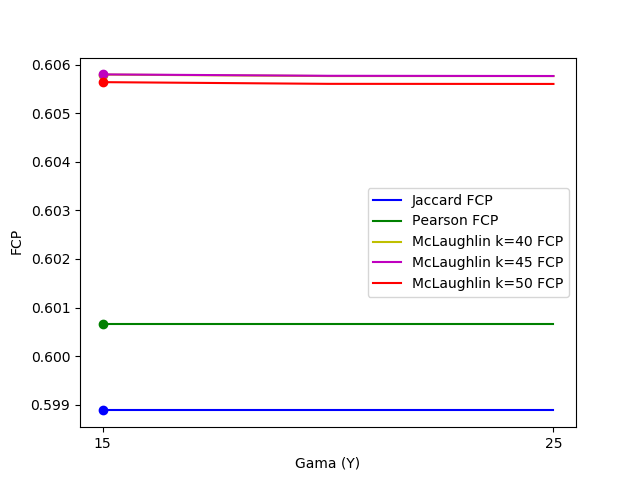
Primero se calcula el número de parejas que concuerdan en el ranking real, para un usuario u, , para todo ítem y , tomando los valores predichos para el usuario y los ítems , y , respectivamente, y de forma similar, los valores reales y .

De la misma forma se calculan las parejas que no concuerdan, :

Con estos dos valores, se suma sobre todos los usuarios.

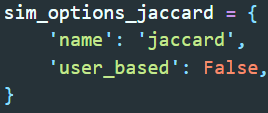
Por último, se calcula la métrica,

 El experimento se realizó con los valores que tuvieron menor error según RMSE, k entre 40 y 50 en intervalos de 5 y gammas entre 15 y 25 en intervalos de 5. Jaccard y Pearson también se incluyeron en la gráfica. Se debe tener en cuenta que FCP, al contrario de RMSE y MAE, se leen al contrario; un valor mayor de FCP significa una mayor precisión.



Efectivamente se invierte el orden, y en este caso Jaccard, que había tenido el menor error consistentemente en todos los experimentos, tiene la menor FCP. McLaughin con k=40 y k=45 tuvieron la misma precisión, y el gamma entre estos valores no afectó la FCP.

1. **Construcción de modelos colaborativos ítem ítem**
2. Para construir el modelo, se utilizó la misma librería que en el caso de filtrado colaborativo usuario usuario. El parámetro al definir el algoritmo *KNNBasic* llamado *sim\_options* puede tomar un valor llamado *user\_based*, de la siguiente manera:



De manera similar, se definió un parámetro *sim\_options\_cosine* y *sim\_options\_pearson*.

1. Predecir la relevancia de los ítems para cada usuario funciona de igual manera para filtrado ítem-ítem que para usuario-usuario y se obtiene el mismo output de la función.

C:\Users\Rogelio Garcia\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\D1D59B0A.tmp

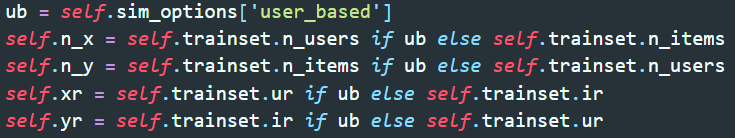
No hubo modificaciones en cuanto a la manera de implementar las tres métricas de similitud, lo único que cambia es el parámetro mencionado en el punto anterior. El parámetro *sim\_options* se define con la opción *user\_based* con valor *False*. Luego se inicializa el algoritmo así (tomando Jaccard como ejemplo):



Luego se llama la función *fit*.



La función *fit* de *KNNBasic* llama la función *fit* de *SymmetricAlgo* define el “x” que utiliza el cálculo de similitudes. Esto hace el papel de transponer la matriz para hacer filtrado por usuarios o por ítems.



El atributo *trainset* es definido en *AlgoBase*, y solamente toma el valor del *train\_set* que fue pasado como parámetro a la función *fit*.

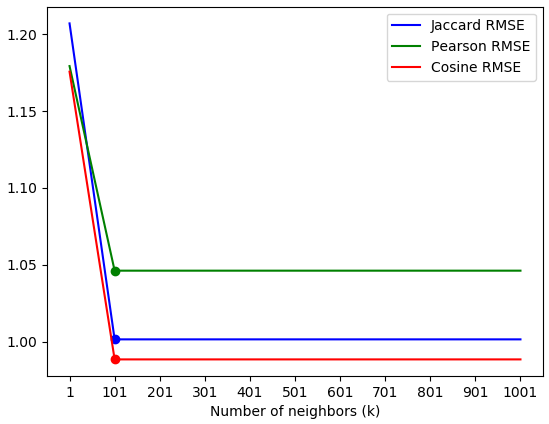
También, al llamar la función *estimate* se utiliza una función *switch* de la clase *SymmetricAlgo*, que intercambia la *x* y la *y* para efectos de estimar las calificaciones.



Estas son las únicas diferencias entre la implementación de filtrado por usuarios y por ítems, dado que son simétricos.

1. Para ser consistentes con la experimentación, también se utilizará RMSE.
2. La estrategia para variar los parámetros es la misma, sin embargo, los valores de k sí cambiaron.

En la siguiente figura se muestran valores para k de entre 1 y 1000. Luego se intentó determinar en qué punto el error deja de variar; este punto cambia mucho cada vez que se ejecuta el script. En la tabla se resumen los valores encontrados tras 10 intentos.



A picture containing screenshot

Description automatically generated

Luego de experimentar con los valores óptimos, se llegó a este resumen:





En general, Jaccard sigue teniendo menos error, aunque la diferencia no es muy grande con coseno. Pearson, que en el filtrado usuario-usuario tuvo una mejor precisión que coseno, en este caso está por debajo. El número óptimo de vecinos para Jaccard, Pearson y coseno son 20, 30 y 40, respectivamente, sin embargo, como siempre después de este número óptimo el error permanece igual, se va a optar por utilizar 30, 40 y 50 como valores óptimos de k.

Con la estrategia de cambiar un threshold, se logró tener el umbral óptimo para cada algoritmo:

A screenshot of a cell phone

Description automatically generated

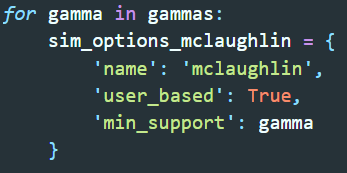
Luego de obtener los parámetros óptimos, se realiza una comparación entre *thresholding* y los k vecinos más similares.

A screenshot of a cell phone

Description automatically generated

En esta ocasión, el algoritmo con menor error es Jaccard con un umbral de similitud de 0,4. Las dos funciones de coseno son muy similares entre ellas, y en general son la segunda mejor opción según error, y Pearson por número de vecinos es menos preciso que Pearson con un umbral de 0,7, pero en general Pearson es la peor opción.

1. McLaughlin, al igual que los otros tres modelos, es un algoritmo *KNNBasic*, y por tanto es simétrico. Por eso, se puede cambiar el mismo valor *user\_based* en el parámetro *sim\_options*. Este parámetro también incluye el gamma, llamado *min\_support*, que será variado para encontrar el gamma óptimo:



El gamma se varió entre 1 y 100, y se graficaron valores de k para McLaughlin de 5 a 30 en intervalos de 5. Consistentemente el valor óptimo de gamma fue 10, y en casi todos los casos McLaughlin sí es una mejora de Pearson, sin embargo, están muy cercanos los valores de sus errores. A partir de k=10, McLaughlin no presenta un cambio en la precisión.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A continuación se muestra una comparación bajo FCP y no RMSE.

A close up of a map

Description automatically generated

Es interesante observar que el valor óptimo para Gama es 1, y que McLaughlin a menor k es más preciso. La interpretación de que gama sea 1 es que siempre se va a tener en cuenta el número de usuarios en común que clasificaron un artista dado, y ese solo hecho hace a McLaughlin más preciso que Jaccard y Pearson, según FCP.

1. **Construcción de la aplicación web**
2. **Análisis de resultados**

…………………………..

Discutir las limitaciones del experimento. No se exploró una combinación k+threshold.

Valdrá la pena inventarse mclaughlin para ítem-item? El paper no lo cubre

Para estas fórmulas, sobre todo mclaughlin, se está asumiendo que el usuario calificó a un artista, sin embargo no es cierto porque solo se basa en el número de veces escuchado. Puede que haya escuchado menos a un artista que realmente sí le gusta.

Idealmente, por razones de tiempo, se evaluaría todo el proceso nuevamente con FCP.

Las razones que lista McLaughlin para que utilizar únicamente Pearson sea mala idea es que puede ser una película difícil de conseguir, “obsure” (whatever that means), o imposible de conseguir. Esto no importa dado que son artistas que sí son accesibles dado que todos hacen parte de la base de datos de lastfm.